LANGLEY GRANT
IN-61-CR
167516
10 P.

Incremental Assessment and Improvement of Software Reliability Midterm Report: NAG-1-824

Larry J. Morell
Department of Computer Science
College of William and Mary
Williamsburg, VA 23185

September 27, 1988

(NASA-CR-183275) INCREMENTAL ASSESSMENT AND IMPROVEMENT OF SOFTWARE RELIABILITY Midterm Report (College of William and Mary) 10 p CSCL 09B
N89-10561
Unclas
G3/61 0167516

Contents

1 Introduction 1

2 A white-box model for assessing fault impact 1

3 Potential Applications 3

3.1 Fault seeding 3

3.2 Estimating the number of remaining faults 4

3.3 Estimating resulting reliability 5

3.4 Confidence bounds of reliability estimates 6

4 Software development status 7

5 Future plans 7

Appendix A Infection and Propagation Analysis: A Fault-Based Approach to Estimating Software Reliability. Submitted to the *11th International Conference on Software Engineering.*

Appendix B Theoretical Insights into Fault-Based Testing. Presented at the *Second Workshop on Software Testing, Verification, and Analysis.*

Appendix C Design Document for GCS Version.

1 Introduction

The objectives of this research are (1) to investigate the relationship between program faults and program failure rates, (2) to develop a method for improved assessment of reliability based upon structural characteristics of the program, and (3) to help design and implement a long-range study of faults in a "realistic" program. Since January we have been pursuing all three of these goals. We have developed a model for analyzing the impact a potential fault may have on a program's failure rate. Based on this model, we have developed a method for estimating the number of remaining faults and resultant reliability, and for determining the confidence bounds in those estimates. We have also developed one version of the software to be used in a multi-version testing experiment.

The specific activities are described in the following sections. Section 2 discusses the model developed for assessing software. Section 3 discusses the potential application areas of the model. Section 4 gives the current status of the Guidance Control System under development. The last section describes future plans. Details may be found in the appendices.

2 A white-box model for assessing fault impact

To develop a structured-based theory of reliability we begin by considering what must happen for a program to fail. Three conditions must be met for a program to fail. First, a fault in the program must be reached. Second, the computation at that point must be affected by the presence of the fault; i.e. some undesirable change to the data state of the program must result. Third, the effect of the modified data state must propagate to an output statement and affect the program's output. Thus, in order to pursue a reliability theory based upon the structure of the code, it seems reasonable to begin with these three notions, the probability of arriving at a point in the code, the probability of a fault affecting the data state at the chosen point, and the probability that the infected state will propagate to the output of the program.

We associate with every point in the program three parameters: its execution rate, its infection rate, and its propagation rate. The *execution rate* of a point in a program is the probability that the point will be executed during an execution selected at random from the environment distribution. Different faults may affect the data states in differing ways and to differing degrees. Similarly one fault may affect distinct data states in differing ways and to differing degrees. Given such situations the best that can be hoped for is a summary statement of how a particular statement will affect the

succeeding data state. Such a summary statement is called the *infection rate* of the statement. The infection rate of a statement S, denoted I_S is the probability that if there were a fault in S, then the succeeding data state would be infected. Note that the infection rate is a function of both the distribution of the data states immediately preceding the statement and the potential faults that might occur at that point. The *propagation rate* of a point in a program is the probability with which viruses in a data state at that point in the program propagate.

One goal is to estimate program reliability given

1. the execution rate of every statement in the program,
2. the infection rate of every statement in the program,
3. the propagation rate of every potential fault point in the program, and
4. a finite partitioning of the program paths.

Let us focus initially on an equivalence class of program paths C. A statement S is in C if and only if S is a member of one of the paths contained in C. By the assumptions above we have E_S, I_S, and P_S for each statement in C. The execution, infection, and propagation rate for a class of paths C are defined as follows. E_C, the execution rate for the class C, is the minimum of all the execution rates of all the statements in C. I_C, the infection rate for the class C, is the minimum of all the infection rates of all the statements in C. P_C, the propagation rate for the class C, is the minimum of all the propagation rates of all the statements in C. The *latent failure rate* for the class it C, denoted by L_C, is defined by:

$$L_C = I_C \cdot P_C.$$

Note that this formula for the latent failure rate is conservative in that it selects the minimum infection rate independent of the minimum propagation rate. In particular these rates may be associated with different statements from the same class.

Program reliability is then estimated in a manner similar to the probabilistic manner used for estimating whether or not a number is prime. Random data is selected from the input space of the equivalence class. The latent failure rate is then interpreted to be a conservative estimate of the probability that if there were a fault in the class then a failure would occur. Thus each successful random execution has a probability of $1 - L_C$ of succeeding. In the simplest case, if K random tests succeed (and none fail), then the estimate of total program reliability is $E_C \cdot 100\%$ with a confidence of

$$1 - [(1 - L_C)]^K$$

This process is repeated for each equivalence class of paths, to obtain a confidence interval for the reliability estimate of the overall program.

Examples of the above analysis can be found in a technical report included in Appendix A. This has been submitted to the 11th Internation Conference on Software Engineering [3]. A paper exploring the theoretical ramifications of fault-based testing upon which the model is founded was presented at the Second Workshop on Testing, Verification, and Analysis in June 1988. This paper is in Appendix B [2].

3 Potential Applications

Infection and propagation analysis can potentially be used to investigate qualities of programs and testing strategies. In particular infection and propagation analysis seems applicable to

1. evaluating a testing strategy
2. predicting the number of remaining faults in a program
3. estimating the reliability of a program
4. providing a confidence bound on an estimate of program reliability

Examples and full development of these ideas found in Appendix A. They are briefly summarized here.

3.1 Fault seeding

Fault seeding, or error seeding as it is frequently known, is the process of inserting artificial faults into a program for the purpose of assessing some quality of the program or of its verification process. Fault seeding is *stratified* if it inserts faults that impact the reliability of the program to pre-specified categories. Most fault-seeding methods produce faults with differing impact on reliability, but are not stratified because the impact is not part of the method. Program mutation is an example of unstratified fault seeding since mutations are not chosen to satisfy a pre-specified impact on reliability.

Infection and propagation analysis can be used to determine the impact on program reliability of faults seeded at a given fault point. This is estimated by multiplying the latent failure rate of the fault seeding point by the execution frequency of that point. To see this, recall that for a fault to cause a program to fail on a given input it must be reached, it must infect the following data state and propagation must occur. The execution frequency is the probability that a point will be reached, and the latent failure rate is an estimate of the probability that the state will be infected and propagation will occur. Thus the product represents the the probability that the fault will cause a failure. To ensure a conservative estimate (i.e., to ensure that the reliability of the program is not overestimated), the category associated with each seeded fault should be as small as possible. For accuracy, the estimate should be no smaller than need be.

A stratified method of seeding can be used to evaluate testing strategies. Faults are seeded with differing impact on reliability. A testing strategy can then be applied, revealing some subset of the seeded faults. This subset thus provides a characterization of the reliability impact the application of that testing strategy has for the program.

3.2 Estimating the number of remaining faults

As originally posed by Mills [1] faults are seeded into programs, which are then tested for a specified period of time. The ratio of revealed seeded faults to total seeded faults is used to estimate the ratio of revealed inherent faults to total inherent faults. Suppose s seeded faults are found and f inherent faults are found during testing. If t_s is the total number of seeded faults and t_f is the total number of inherent faults then the presumption is made that:

$$\frac{f}{s} \approx \frac{t_f}{t_s}$$

and hence

$$t_f \approx \frac{f \cdot t_s}{s}.$$

As pointed out by Mills[1] this approach to estimating remaining faults rests upon the strong assumption that the seeded faults have the same probability of discovery as those already present in the program. Mills called this "a nontrivial program-theoretic problem." We feel that infection and propagation analysis is a useful approach to solving this problem.

We begin with a categorization of the impact a fault has on reliability. Let $r_1, r_2, ... r_k$ be k different failure rates. The intent is that each r_i be significantly different than the other rates, say, by an order of magnitude. Each fault contributes a certain percentage to the overall failure rate of the program; each fault can therefore be categorized into one of the k categories, determined by the r_i that most closely bounds its failure rate. t_s faults are now seeded into the program, with t_{s_i} faults seeded with reliability impact r_i. Thus,

$$t_s = \sum_{i=1}^{k} t_{s_i}$$

Also, assuming there are t_f inherent faults and t_{f_i} inherent faults of reliability impact r_i, we have

$$t_f = \sum_{i=1}^{k} t_{f_i}$$

Testing then proceeds as usual finding s seeded faults and f actual faults. Let s_i denote the number of seeded faults found with reliability impact r_i. Let f_i denote the number of actual faults found with reliability impact r_i. Thus

$$f = \sum_{i=1}^{k} f_i$$

and

$$s = \sum_{i=1}^{k} s_i.$$

The estimate of the number of remaining faults is

$$t_f = \sum_{i=1}^{k} t_{f_i} \approx \sum_{i=1}^{k} \frac{f_i}{s_i} \cdot t_{s_i}$$

3.3 Estimating resulting reliability

We can now estimate the resulting reliability. We have:

$$\frac{f_i}{s_i} \approx \frac{t_{f_i}}{t_{s_i}}$$

and thus for all i from 1 to k,

$$t_{f_i} \approx \frac{f_i}{s_i} \cdot t_{s_i}$$

Thus the maximum impact the remaining faults (those not yet uncovered) can have on the reliability is

$$\sum_{i=1}^{k} (t_{f_i} - f_i) \cdot r_i.$$

Hence the reliability is at least

$$1 - \sum_{i=1}^{k} (t_{f_i} - f_i) \cdot r_i.$$

3.4 Confidence bounds of reliability estimates

We begin by partitioning all program paths into finitely many distinct path classes called *trips*. This can be accomplished by describing the program flow graph as a regular expression r where each of the r's is of the form $r_1 + r_2 + \cdots + r_n$, and each of the r_i's represents a trip. Define the *propagation rate* for a trip to be the minimum propagation rate for all statements of the trip. Similarly, define the *infection rate* for the trip to be the minimum infection rate for all statements of the trip. Finally, let the *latent failure rate* for a trip be the product of the infection and propagation rates for the trip.

Given an initial estimate of reliability, the confidence in this estimate can be determined as follows. The latent failure rate is interpreted as the minimum frequency of program failure that would be expected if there were any faults along the associated trip. Reliability is then estimated using the techniques of the previous section by some other means, e.g., direct estimation. The key issue is, how accurate is this estimate?

Consider the situation in which there are n trips in the program, $t_1, t_2, \cdots, t_n$ with latent failure rates of $l_1, l_2, \cdots, l_n$. Select the trip with the largest l_i. Since each execution of a path in the trip has at least a probability l_i of failing on a random execution, the

probability of N successive successful executions of paths in the trip is $[1 - l_i]^N$ if there were a fault along one of the paths in the trip. Hence

$$1 - [1 - l_i]^N$$

is the probability that at least one failure would occur in N executions. Thus, this formula can be considered as the confidence in the assertion that there is no fault in the designated trip. Hence it can be concluded that the reliability of the program is at least the trip execution rate for trip i, E_{t_i}, with the confidence given as above.

This process may be repeated for the trip with the next lowest l_i value, and so on until the accumulated reliability is as close as desired to the experienced reliability. At this point the estimated reliability can be accepted as the actual reliability of the program within the specified confidence.

4 Software development status

William and Mary is collaborating with other institutions in the development of several versions of landing software called the Guidance Control System. Most of the spring was spent analyzing the specification. In late spring the implementor, Jeff Voas, attended an orientation seminar at RTI in North Carolina to learn the mechanics of posing questions. During the summer he completed a software design and wrote the code. Appendix C contains a current copy of the design document and the code (which compiles error-free).

5 Future plans

Propagation and infection analysis will continue be be explored as a basis for evaluating verification methods, with a particular emphasis on reliability assessment. We are exploring the use of latent failure rates as a basis for a *testing complexity metric*. We speculate that more testing is required for programs with lower latent failure rates. We further speculate that programs with high latent failure rates are can be sufficiently tested by statement testing, but those with lower latent failure rates will require more extensive coverage. This seems reasonable because high latency implies almost every execution will reveal a fault if there is one present, hence statement coverage might be sufficient. A low latency rate implies that many locations exist within the code where faults could be seeded an remain undetected, thus more extensive testing may be required.

A design review must now be conducted on the GCS and testing must begin on the modules of the program. Plans are currently being developed to conduct this testing.

References

[1] H. D. Mills. On the statistical validation of computer programs. *IBM Federal Systems Division, Report FSC-72-6015, Gaithersburg, MD.*, 1972.

[2] L. J. Morell. Theoretical insights into fault-based testing. *Second Workshop on Software Testing, Validation, and Analysis*, pp. 45–62, July 1988.

[3] L. J. Morell and J. M. Voas. Infection and propagation analysis: a fault-based approach to estimating software reliability. *Submitted to the 11th International Conference on Software Engineering*, 1989.